DOCUMENT RESUME

ED 084 821 EM 011 640

AUTHOR Hayes, Joan; Kheriaty, Larry
TITLE The Enhancement of Interactive Computing for CAI Users Through Development of a Multilingual Interpreter.
INSTITUTION Western Washington State Coll., Bellingham.
PUB DATE Aug 73
NOTE 6p.; Paper presented at the Summer Meeting of the Association for the Development of Instructional Systems (Ann Arbor, Michigan, August 1973)

EDRS PRICE MF-$0.65 HC-$3.29
DESCRIPTORS Computer Assisted Instruction; *Computer Programs; Computer Science; Higher Education; *Machine Translation; *Man Machine Systems; Program Descriptions; *Programing; *Programing Languages
IDENTIFIERS APL; Front End Processors; Generalized Interpreter; *Multilingual Interpreter; Scanning

ABSTRACT

The Multilingual Interpreter (MI), a language processor capable of executing more than one language, is being used at Western Washington State College to enhance interactive computing for computer-assisted instructional (CAI) users. The MI's two major functions are, first, that of scanning and translating a source language into internal interpretive code, accomplished by programs called front-end processors, and, second, that of interpreting, or executing the internal code, accomplished by a program called the generalized interpreter. As a result, unsophisticated users are equipped with an easy-to-learn language whose power can continue to satisfy them as their needs increase in complexity. APL is also provided despite the fact that the available core is limited to 84k. (PB)

FILMED FROM BEST AVAILABLE COPY

ED 034821

The Enhancement of Interactive Computing for CAI Users Through Development of a Multilingual Interpreter.

Joan Hayes and Larry Kheriaty

Western Washington State College

U.S. DEPARTMENT OF HEALTH, EDUCATION & WELFARE
NATIONAL INSTITUTE OF EDUCATION
THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL NATIONAL INSTITUTE OF EDUCATION POSITION OR POLICY

Presented at the Summer Meeting,

Association for the Development of

Instructional Systems

Ann Arbor, Michigan

August, 1973

EM 011 640

I am another person who was particularly taken by Jess Heines' essay in the July ADCIS Newsletter, entitled "What's In It For Us?" His view that "the production of instructionally-oriented computers for education takes a distinct back seat in the industry's production line" is shared by other teachers, many of whom are involved in post-secondary education. Few instructors are interested in instructional technology which would relieve them of all their teaching responsibilities, but they would like to have available short units or problem-solving projects which take advantage of the instructional capabilities of the computer.

Good teachers who are inexperienced programmers would like to get beyond the fun-n-game level Mr. Heine decries; most have particular material they would like to cover in their own teaching style. So the first question is:

1) How do you provide an easy-to-learn language whose power will continue to satisfy users as their sophistication increases?

2) Most Computer Center staff members and non-timid programmers would like to have a language such as APL. That brings us to the second question: How do you provide APL if the available core is limited? On our campus, a 256K Model 40 (supplemented by a 7090) provides the total computing power required for the instructional and administrative data processing needs of 9000 students.

3) The third consideration is the ubiquitous problem-solver. Life in a terminal room would be far less creative and interesting without him, but he, too, requires a language similar to APL.

Western Washington State College is a 4-year college struggling to keep creative fires alive despite the quenching effects imposed by financial facts of life.

We have found a solution which is working well for us, and in 84K. It has been and is a stable system which becomes more available to users with each modification.

In 1970, we began with an interactive PL/I, a subset called WPL which could be run in an 88K partition of the 360/40. In 1970, also, we were running COURSEWRITER III, but users had to choose between signing on for problem solving (PL/I) in the afternoon, or for CAI (COURSEWRITER III) in the morning.

In 1973, the language authors decided to combine COURSEWRITER III with the problem-solving language by front-ending a syntax scanner for COURSEWRITER statements into the PL/I interpreter. At that point, the interpreter became MULTILINGUAL. That marriage was successful enough to provide encouragement for the addition of a BASIC front end processor in March, 1973, a relationship which so far has remained companionable.

How is this accomplished? And what does it mean to the user?

The WWSC terminal system has two levels of generality. The Terminal Control System or telecommunications monitor, (TCS) is configured in such a way that any program written in assembly language can be brought up with relative ease. The requirements for running a program as a TCS application are simple:

First, the program must use the TCS-provided macros for I/O both to disk and to the terminal. These macros minimize the programmer's effort for doing I/O.

Second, the program must reside in object form on the TCS application library. Any application program can be invoked from the terminal by a %EXEC command. The user at the terminal can use the %EXEC command at any time to go from one application to another. Applications we use regularly are:

WPL
BASIC
RPS - for file handling
CAISCAN - for CAI diagnostics
REGISTRATION

From time to time we also write special application programs to solve some particular interactive computing need.

The second level of generality centers around the MULTILINGUAL INTERPRETER, the language processor capable of executing more than one language. The operation of the interpreter can be divided into two functions:

First That of scanning and translating a source language into internal interpretive code, accomplished by programs called "front-end processors".

Second That of interpreting, or executing the internal code, accomplished by a program called the "generalized interpreter", referred to earlier.

The fact that the scanning or compiling operation is separated from the interpreting or execution operation has several advantages:

a) To add a different language, only the syntax scanner for that language need be written. BASIC, for example, was added in about 20% of the time required to write the original interpreter.
b) Languages can be mixed to suit the type of problem being solved.
c) Time can be saved by storing programs in compiled form.
d) Programs may be run in source form during debugging and in compiled form when in production.

The BASIC and PL/I front ends both produce the same type of code. It would be feasible to write front end processors for other languages such as FORTRAN or APL, or, eventually maybe a subset of PLANIT. Users, then, do not need to feel locked-in in any sense, to a single language, or to separate languages, with separate system overhead requirements.

Does this really mean anything for the neophyte interested in instructional uses of the computer? We think so, even as he becomes a non-neophyte.

COURSEWRITER III, with its OP codes, automatic sequencing, provision for branching, and its answer-processing capabilities, is recognized among other things as a good "beginning" language for non-programmers interested in CAI.

But when the once-neophyte becomes more sophisticated, he is likely to request a capability non-extant in COURSEWRITER III, and available only by writing a program in 360 Assembler Language. Now he may have made progress, but chances are remote he's made <u>that</u> much progress; unless the CAI Center has a large staff for writing custom-built programs, and we do not, a budding author is frustrated.

At WWSC, "functions" have been replaced by PL/I programs which can be CALLED from the main program. (Hopefully, the CAI author has been picking up some PL/I along the way.) Or, if remedial material or any series of statements is used frequently, the statements can be placed in a segment and invoked with a LINK from different parts of the course. When the called segment is complete, control returns to the statement after the LINK. If a routine has a generalized use, it is placed on a public file, and can be called from any terminal.

Another CW3 op code provided in addition to LINK is XCTL, which can be used to chain to another segment at the same link-level.

EXIT op code adds a return operation to the course segment.

(Since we recommend writing courses in relatively short modular units, it is sometimes easiest to LINK to a major segment which is divided into sub-units which XCTL from one to another, and EXITs back to the main course.)

Because of the close relationship between PL/I and CW3 front ends, the CAI programmer can place PL/I statements anywhere within a course. With PL/I statements, the author can access all the automatically provided data areas of CW3; possible because counters and time counters are stored as arrays, but buffers, return registers, parameters and switches are stored as character strings. Users have gratifying control over the dimensions of the variables and can create their own - either for permanent inclusion or as a scratch pad while working with a program.

There's another advantage to the multilingual arrangement so far as testing for numeric matches is concerned. By coding the 'X' modifier on an answer statement, the student answer and the ca are treated as PL/I expressions, and numeric responses compared for equivalence. (So, 5.0 and sqrt (25) would both be match for a ca of 5, where the x-modifier is used.)

A relatively unusual capability we are finding more and more useful is that of being able to dynamically construct a PL/I statement in a buffer during execution, and pass that statement to the PL/I front end processor both translation and execution. As an example - Suppose we were writing a math course, and talking about a certain type of formula. It is possible to have the student type in the formula, and to have the formula executed.

Or if we were working with a programming course, a student could type in a statement, have it executed, and observe the results.

If desired, the author can restrict the amount of information available to the student through use of parameters delimiting the calc command. The student can have access to:

1) all counters, buffers, etc.,
2) restricted variables in the program,
3) his own variables,
4) no variables, but have expression evaluation,
5) no calc command at all.

We have been particularly pleased at the ease with which supervisory functions on the system are handled - namely, there are none. Authors housekeep their own datasets with standard utilities executed from the terminal (compress, backup). Authors type to a temporary copy which is stored and becomes permanent.

Constraints on the system are a result, primarily of the limited core. We could have an extended BASIC if the system were larger, for example. (Actually, BASIC is mostly a spooker since the original two, if expanded, could do more than BASIC.) We are just beginning to add new BASIC units to our curriculum, and some of those would be more easily translated if there were more space to work in.

Well - you can guess that we might not be completely satisfied with the system as it stands, but we do admit to near-bliss with our instructional menage a trois.